Manuel d'utilisation de l'environnement Linux pour le Concours de programmation UQAM

Hiver 2003

1 Pour vous connecter à la machine

Les machines sont déconnectées du réseau et ont toutes la même configuration de départ décrite dans les pages qui suivent. Pour vous connecter à la machine qui vous a été assignée (PK-S570 : `e03--e26` ou PK-S560 : `f03--f26`), vous devez tout d'abord taper le nom d'usager suivant :

```
salle
```

Vous devez ensuite indiquer le mot de passe suivant :

```
labunix
```

Vous serez alors connecté au compte de ce poste.

2 Soumission d'une solution

Pour soumettre une solution à un problème, disons le problème `X`, vous devez créer un *exécutable* en suivant les directives indiquées à la section 5. Cet exécutable doit nécessairement s'appeler "`X`" — sauf pour Java où un fichier `X.jar` doit plutôt être créé.

Une fois l'exécutable `X` (ou `X.jar`) créé, vous pouvez le soumettre pour évaluation en exécutant simplement la commande suivante (même dans le cas de Java) :

```
rendre_probleme X
```

Le fichier exécutable `X` (ou `X.jar` pour Java) doit être dans le répertoire courant. Le script `rendre_probleme` transmettra tous les fichiers du répertoire courant nommés `X` ou `X.*` (donc le fichier `X.jar` pour une solution Java, où les fichiers `X` et `X.pl` pour une solution Prolog).

Après quelques secondes, vous recevrez une réponse qui vous indiquera si votre solution a été jugée correcte ou non.

N'oubliez pas que si vous soumettez une solution incorrecte au problème, vous avez le droit de resoumettre une autre solution au même problème un peu plus tard. Toutefois, chaque solution incorrecte ajoute 20 points (minutes) de pénalité au temps associé à une solution finale correcte.

Note importante : Durant la période *de pratique*, il est important que vous exécutiez au moins une fois la commande `rendre_probleme` (même si vous savez que votre solution n'est *pas* correcte), de façon à vérifier que votre machine est bien branchée au réseau et correctement identifiée dans la base de données.

3 Impression d'un fichier

Pour imprimer un fichier, les commandes standards ne peuvent pas être utilisées, puisque votre machine est déconnectée du réseau. Vous devez plutôt utiliser la commande suivante :

`imprimer` *nom_de_fichier*

Notez que cette commande ne permet d'imprimer *qu'un seul* fichier à la fois.

4 Environnement Linux et coquille `bash`

4.1 Commandes Unix de base

`ls`	Donne la liste des fichiers dans le répertoire courant.
`pwd`	Affiche le répertoire courant.
`cd`	Change le répertoire courant.
`mkdir`	Crée un nouveau répertoire.
`cat`	Affiche le contenu d'un fichier.
`less`	Affiche, page par page, le contenu d'un fichier. `more` peut aussi être utilisé.
`rm`	Supprime un fichier.
`cp`	Copie un fichier (d'une source vers une destination).
`mv`	Déplace (renomme) un fichier (source vers destination).
`xterm &`	Crée une nouvelle fenêtre et sa coquille.
`diff`	Compare deux fichiers (ligne par ligne) et affiche les différences (`diff -w` permet de comparer en ignorant les blancs).
`chmod`	Modifie les permissions d'un fichier.

Figure 1: Commandes Unix de base

Comme il s'agit d'un environnement Linux, les commandes standard Unix sont donc disponibles (voir Figure 1).

Suggestion : Au début du concours, créez un répertoire distinct pour chacun des problèmes, par exemple :

```
mkdir P
mkdir Q
```

Ensuite, pour chacun des problèmes, assurez vous de conserver tous les fichiers associés à ce problème dans ce répertoire.

4.2 Éditeurs de texte

Les éditeurs de texte suivants sont disponibles : `gedit`, `pico`, `vi`, `emacs`.

Pour la plupart des langages, `emacs` définit un mode d'édition associé au langage. À l'ouverture (ou à la création) d'un fichier avec l'extension appropriée, `emacs` se mettra automatiquement dans le mode supportant ce langage :

Ada	`.ads` ou `.adb`
C	`.c`
C++	`.cpp` ou `.c++`
Haskell	`.hs`
Java	`.java`
Prolog	`.pl`
Perl	`.pm`

4.3 Historique des commandes

La coquille `bash` permet de préserver un historique des commandes exécutées :

- `history` : Affiche la liste des commandes exécutées précédemment.
- `!`n : Réexécute la commande numéro n (telle qu'indiquée par `history`) de l'historique.
- `!xyz` : Réexécute la commande qui débutait par `xyz`.
- `Tab` : Complète la commande ou nom de fichier partiellement tapée. Affiche les différentes possibilités si la complétion n'est pas unique.
- *Flèche vers le haut* : Affiche la commande précédente de l'historique ; il suffit alors de taper `Return` pour que la commande s'exécute. La *Flèche vers le bas* parcourt l'historique en sens inverse.

5 Génération d'un exécutable

Dans ce qui suit, on suppose qu'on veut générer un exécutable pour le problème `X` et que cet exécutable doit donc s'appeler "`X`".

Rappelons qu'un exécutable doit avoir des permissions d'exécution. La commande `ls -l` permet d'afficher les permissions, par exemple :

```
% ls -l X
-rwxrwxr--    1 e01 e01        2340 mar 22 06:21 X
```

Pour ajouter les permissions d'exécution à un fichier `X`, il suffit de taper la commande suivante — le `+x` signifie d'ajouter la permission d'exécution :

```
chmod +x X
```

Une fois les permissions d'exécution ajoutées, il suffit ensuite de taper la commande suivante pour exécuter le programme `X` :

```
./X
```

5.1 Ada

Pour compiler un programme Ada95 `X.adb` (idem pour `X.ads`), vous devez utiliser la commande suivante :

```
gnatmake X.adb
```

Note : Contrairement à ObjectAda un programme doit être dans un fichier avec extension `.adb` *et non* avec `.ada`.
Note : il est possible que l'éditeur de lien génère un avertissement du style suivant :

```
gnatlink --GCC=/usr/gnat/bin/gcc X.ali
/usr/gnat/lib/gcc-lib/i686-pc-linux-gnu/2.8.1/adalib/libgnat.a(a-adaint.o): In function '__gnat_tmp_name':
a-adaint.o(.text+0x504): the use of 'tmpnam' is dangerous, better use 'mkstemp'
```

Cet avertissement peut simplement être ignoré.

5.2 C/C++

Pour compiler un programme `X.c`, vous utilisez l'une des commandes suivantes, selon que vous utilisez `C` ou `C++` :

```
gcc -o X X.c

g++ -o X X.cpp
```

5.3 Haskell

Il faut définir, dans le fichier `X.hs` approprié, un module `Main` et dans ce module la fonction `main` :

```
module Main where

main =
    ...
```

Ceci constituera le point d'entrée de l'exécutable.

Une fois les fonctions mises au point et codées dans ce fichier `X.hs`, il faut ajouter en tête du fichier la ligne suivante pour créer l'exécutable :

```
#!/usr/bin/runhugs
```

Vous pouvez ensuite copier votre fichier `X.hs` dans un fichier `X` et mettre les permissions de ce dernier fichier en mode exécution.

5.4 Java

Pour Java, vous devez créer un fichier `jar` de la façon suivante, en supposant que votre classe principale est dans le fichier `X.java` :

1. Écrire le programme Java sans utiliser les "*packages*". (Ne pas écrire "`package ...`" au début des classes).

2. Sauvegarder les classes dans un répertoire et nommer la classe principale "`X`" (sinon indiquer le nom approprié dans les commandes et le fichier `Manifest.txt` ci-bas).

3. Aller dans le répertoire ci-haut, et exécuter la commande suivante:

   ```
   javac X.java
   ```

4. Si tout compile sans problème, créer (dans le même répertoire) un fichier appelé `Manifest.txt` dont le contenu est exactement les deux lignes suivantes (les deux terminées par un retour de ligne) :

   ```
   Manifest-Version: 1.0
   Main-Class: X
   ```

5. Créer le fichier `jar` exécutable en exécutant la commande suivante (toujours dans le même répertoire, et en faisant attention à l'ordre des paramètres):

   ```
   jar -cmf Manifest.txt X.jar *.class
   ```

6. Tester cet exécutable en tapant (toujours dans le même répertoire):

   ```
   java -jar X.jar
   ```

5.5 Perl

Il faut mettre la ligne suivante au tout début du fichier (première ligne), nommer le fichier `X`, et ajouter la permission d'exécution avec `chmod` :

```
#!/usr/bin/perl
```

5.6 Prolog

Pour produire un exécutable pour un programme `X.pl`, il faut définir le prédicat suivant dans votre programme :

```
runtime_entry(start) :-
  ...
```

Ce prédicat constituera le point d'entrée de votre exécutable (le but à résoudre).

Une fois les prédicats mis au point et codés dans le fichier approprié il faut utiliser la commande suivante pour créer l'exécutable :

```
spld -o X X.pl
```

L'exécutable aura alors comme nom "`X`". Il reste ensuite à modifier les permissions pour ajouter le mode exécution.

6 Tests de vos programmes

Pour tester un programme avant de le soumettre pour évaluation, la stratégie de base est la suivante :

- Créez un (ou plusieurs) fichier(s) de données (en vous inspirant des exemples donnés dans les énoncés de problème), par exemple, `X.data`.
- Pour chaque fichier de données, créez un fichier contenant le résultat attendu, par exemple, `X.attendu`.
- Compilez et générez l'exécutable `X`.
- Exécutez votr programme (sauf pour Java, où vous devez utiliser "`java -jar X.jar`") et comparez les résultats obtenus avec ceux attendus :

```
./X <X.data >X.resultat
diff X.attendu X.resultat
```

Si des différences sont signalées par `diff`, alors votre solution n'est pas correcte, en supposant évidemment que vos données et les résultats attendus sont corrects (vous pouvez aussi utiliser "`diff -w`" pour ignorer les espaces). Si aucune différence n'a été signalée et que vous croyez que vos tests sont complets, vous soumettez ensuite votre solution :

```
rendre_probleme X
```

Il est important de noter que c'est une approche semblable (avec `diff` ou `cmp`), mais avec possiblement plusieurs jeux d'essai, qui est utilisée par le script de soumission et vérification (`rendre_probleme`).

7 Consultation du score des différentes équipes

Durant les trois (3) premières heures du concours, vous pourrez consulter l'un des juges (machine à l'avant de la salle) pour voir sur son écran l'évolution des scores des différentes équipes. Toutefois, de façon à conserver le suspense final, les scores ne seront pas divulgués durant la dernière heure du concours `;)`